08-语言的模型：如何打破单一语言局限，让设计更好地落地？

你好！我是郑晔。

经过前面几讲，我们已经学习了如何去了解一个现有软件的设计。从这一讲开始，我们就进入到新的模块，讨论如何设计一个软件。做设计之前，我们要先知道手边有哪些工具。所以在这个模块开启之初，我们先来讨论程序设计语言。

或许你会觉得，程序设计语言有啥好讨论的？哪个程序员没有一门看家的程序设计语言呢？不知道你是否遇到过这样的问题：

- 面向对象用来组织程序是好，但我用的是C语言；
- 我用的是C++，函数式编程的好，跟我有什么关系；
- 动态语言那些特性很好，可惜我用的是Java；
- ……

如果你这么想，说明你被自己的看家本事给局限住了，这种思维方式会让你即便学到了更多的好东西，也只能无可奈何。

其实，程序设计语言之间没有那么泾渭分明的界限，程序员唯有多学习几门语言，才能打破语言的局限，让设计更好地落地。你可以根据项目特点选择合适的语言，也可以将其它语言一些优秀的地方借鉴过来。Andrew Hunt和David Thomas在《程序员修炼之道》（The Pragmatic Programmer）中给程序员们提了一项重要的建议：**每年至少学习一门新语言。**

可是，语言那么多，我要一个一个都学过去吗？学语言到底在学什么呢？

其实，程序设计语言本身也是一个软件，它也包含模型、接口和实现。而**我们学习程序设计语言主要是为了学习程序设计语言提供的编程模型**，比如：不同的程序组织方式，不同的控制结构等等。因为不同的编程模型会带给你不同的思考方式。

既然要学习编程模型，我们就要先知道编程模型设计的来龙去脉，所以，今天我先带你领略一下程序设计语言的发展历程。

程序设计语言发展简史

我们今天接触到的程序设计语言都是图灵完备的。这里的“图灵完备”指的是语言指定的数据操作规则能够实现图灵机的全部功能（图灵机的概念是由阿兰·图灵提出的，图灵机为计算机能够解决的问题划定了一个边界）。所以，图灵机是所有程序设计语言最底层的模型，程序设计语言都是在这个基础上生长出来的，包括众所周知的计算机基础：用0和1编码。

我们今天的计算机能够识别的都是0和1，但真正用0和1直接写代码的人少之又少，因为实在太麻烦了。所以，早在计算机诞生之初，就产生了**汇编语言**，它可以将那些0101的操作符变成更容易记住的ADD、MOV之类的指令。

相比于01串，汇编虽然进步了一些，但人们很快就发现，用汇编写程序也是非常痛苦的事情，因为只有对计算机了如指掌，才能写好汇编。更可怕的是，即便你熟练掌握了一种计算机的汇编语言，换成另外一种计算

机，你也必须从头学过。

这时，就轮到**高级程序设计语言**登场了。

第一门被广泛使用的高级程序设计语言是Fortran，它为程序设计语言的发展奠定了基础。比如，一些基本控制结构出现了，数据开始拥有了类型（类型就是一种对内存数据的解释方式）。虽然这些东西在今天看来非常简单，但和那个年代使用的汇编相比，简直是一个巨大的飞跃。

Fortran对于计算机的发展起到了巨大的推动作用，人们也逐渐认识到高级程序设计语言对于开发效率的提高。接下来，人们开发了各种高级程序设计语言，不断地探索怎样写好程序。

早期程序设计语言探索的集大成者就是**C语言**，它提供了对于计算机而言最为恰当的抽象，屏蔽了计算机硬件的诸多细节。时至今日，C语言依然受众广泛。

随着高级程序设计语言的发展，门槛逐步降低，人们可以开发的程序规模也逐渐膨胀。这时候，**如何组织程序**成了新的挑战。有一种语言搭着C语言的便车将面向对象的程序设计风格带入了主流视野，这就是C++。很长一段时间内，C++成为了行业中最主流的选项，既兼容C语言，又提供了很好的程序组织方式。

虽然各种高级程序设计语言已经屏蔽了很多细节，但有一个问题始终没有得到很好的解决，也由此引发了更多的问题，这就是**内存管理**。其实，人们早就在尝试各种屏蔽内存管理的方式，但因为早期计算机硬件性能有限，所以没有任何一种方式能够成为行业主流。

后来，计算机硬件的能力得到了大幅度提升，这让那些在故纸堆里的技术又焕发了新的活力。这个阶段的胜利者是Java，一方面，它支持面向对象编程；另一方面，它还有垃圾回收机制——一种内存管理的方式。

Java的路其实也很坎坷，因为它早期在个人电脑上的尝试并不算成功。后来选择了企业级开发的赛道，才有机会展现自己的优势。因为企业级服务器本身性能优于个人电脑，对Java有更高的容忍度，它才得到了机会，不断进行自身的优化。

当硬件不再是程序设计语言的发展障碍之后，程序设计语言又该如何发展呢？

从前面的历程不难看出，程序设计语言的发展就是一个“逐步远离计算机硬件，向着待解决的问题靠近”的过程。所以，程序设计语言接下来的发展方向就是**探索怎么更好地解决问题**了。

前面说的这些只是程序设计语言发展的主流路径，其实还有一条不那么主流的路径也一直在发展，就是**函数式编程的程序设计语言**，这方面的代表就是LISP。

在这条路上，刚开始，很多人都是偏学术风格的，他们更关心的是解决方案是否优雅，也就是说，如何解决问题，如何一层层构建抽象。他们也探索更多的可能，垃圾回收机制就是从这里来的。但同样受限于当时硬件的性能，这条路上的探索在很长一段时间之内都只是一个小众游戏。

当硬件的性能不再成为阻碍，如何解决问题开始变得越来越重要时，函数式编程终于和程序设计语言发展的主流汇合了。促进函数式编程引起广泛重视也还有一个硬件因素：**多核**。

多核的出现，本身是IT行业应对CPU发展进入瓶颈期的一个解决方案，但它却打破了很多程序员只习惯于利用一个CPU写程序的传统方式。

为了利用多核的优势，人们探索了各种方案，今天看到的各种并发模型、异步模型等解决方案都从那时开始得到了蓬勃的发展。函数式编程在这个方面的探索就是利用自己声明式的表达方式屏蔽了硬件差异。让人们注意到函数式编程的价值的就是著名的MapReduce。

函数式编程的兴起，让那些在函数式编程社区的探索随之兴起，比如，声明式编程、DSL、元编程等等。一些后出现的程序设计语言开始将面向对象和函数式编程二者融合起来，比如Scala。而像Java和C++这些“老战士”则逐渐地将函数式编程的支持加入到语言之中。

相比于这些“正规军”，还有一股力量也逐渐从边缘走上了舞台，这就是**动态语言**，代表语言有 Perl、Python、Ruby、PHP等等。以前，人们更喜欢用“脚本语言”称呼这类程序设计语言，这个名字表明，它就是为了简单地解决一些特定的问题而出现的。所以，在人们心目中，它们显得并不那么正式。但它们简单、轻巧的特性有效地降低了入门的门槛，也赢得了一大批拥趸。

语言的发展就是一个互相学习和借鉴的过程。以前，动态语言的弱项在于不适用于规模比较大的工程，而近些年来，随着动态语言用户的增多，配套的工具也逐渐多了起来，动态语言项目的规模也逐渐增大。而在主航道的程序设计语言，也纷纷向动态语言学习，努力地简化代码编写的难度，比如，Java和C++都开始支持类型推演（Type Inference），目的就是让程序员少敲几个字符。

至此，我简单地带你回顾了一下程序设计语言的发展历程，梳理了程序设计语言的发展脉络。从中不难看出，如果把程序设计语言当作一个软件，它的发展历程就是一个逐渐添加新模型的过程，而其发展的结果就是如今的开发门槛越来越低，能够开发的程序规模越来越大。

一切语法都是语法糖

现在，你已经能更好地理解我们在前面提出的说法，**学习程序设计语言其实就是要学习语言提供的编程模型。**

以我学过的一些程序设计语言为例：

- C语言提供了对汇编指令直接的封装。
- C++先是提供了面向对象，后来又提供了泛型编程。
- Java把内存管理从开发者面前去掉了，后来引入的Annotation可以进行声明式编程。
- Ruby提供了动态类型，以及由Ruby on Rails引导出的DSL风格。
- Scala和Clojure提供了函数式编程。
- Rust提供了新的内存管理方式，而Libra提供的Move语言则把它进一步抽象成了资源的概念。

既然学习新的程序设计语言是为了学习新的编程模型，反过来也可以说，**不提供新编程模型的语言是不值得刻意学习的**。如果你已经学会了一两门程序设计语言，学习一门新的语言其实并不困难，因为每种语言提供的新模型是有限的，基本的元素是类似的，无非是用了不同的关键字。

所以，学习新语言，只是在做增量的学习，思维负担并没有那么沉重。一旦对于程序设计语言的模型有了新的认识，你就能理解一件事：**一切语法都是语法糖**。

语法糖（Syntactic sugar）是英国计算机科学家彼得·兰丁发明的一个术语，指的是那些为了方便程序员使用的语法，它对语言的功能没有影响。

懂得了语法糖的道理，要想更好地理解程序设计语言，一种好的做法就是打开语法糖，了解一下语法是怎么实现的：

- 类型是一种对内存的解释方式。
- class/struct是把有相关性的数据存放到一起的一种数据组织方式。
- Groovy、Scala、Kotlin、Clojure等JVM上的新语言，提供了一种不同于Java的封装JVM的方式。
- ……

通过前面的介绍，你也看到了，语言的发展并非一蹴而就，而是一个渐进式的发展历程。一些新的尝试总会在一些不起眼的地方冒出来，而且语言之间也在相互借鉴。

如果你能每年学习一门新语言，起初，你可以了解不同的编程模型。当你的积累足够多了，学习语言就是在跟踪程序设计语言的最新发展了。

当你手里有了足够多的“武器”时，你就可以打开思路，运用不同的方式解决问题了，甚至把其它语言的好东西，借鉴到自己使用的语言中。

总结时刻

今天，我们谈到了程序设计语言。学习不同的程序设计语言可以帮助我们更好地落地设计，也可以让我们向不同的语言借鉴优秀的方面。

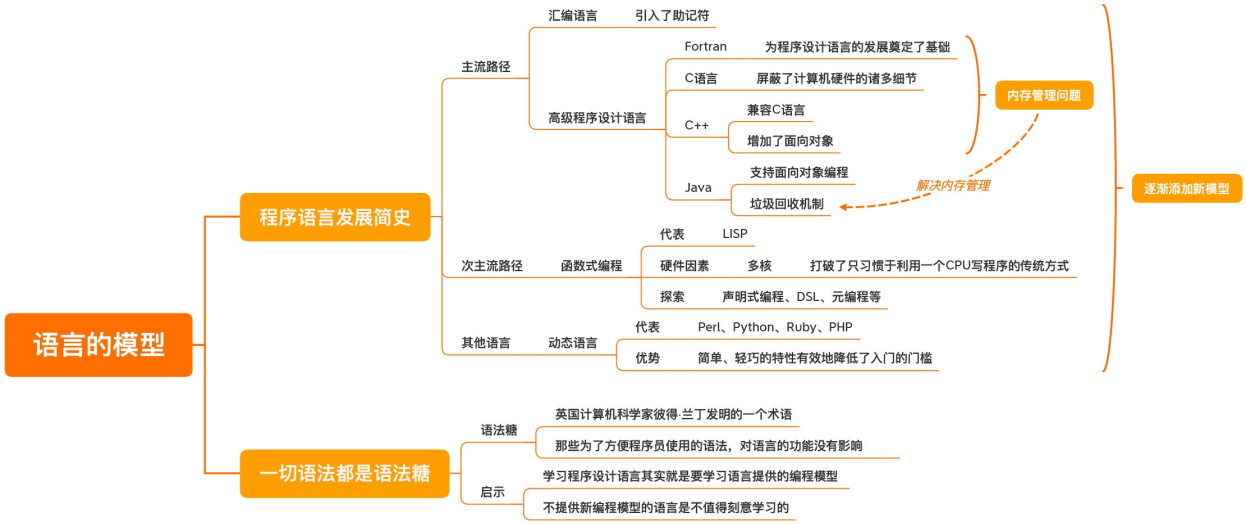
我们简要地了解了程序设计语言的发展历史，从最开始的对机器模型的封装，到今天不断降低的开发门槛，程序设计语言的演化从未停止。我们也看到各种不同的编程风格在经历了最初各自独立的发展之后，开始慢慢融合。

对程序设计语言发展的了解，可以帮助我们理解一件事：**一切语法都是语法糖**。新的语法通常是在既有的结构上不断添加出来的，为的是简化代码的编写。

《程序员修炼之道》鼓励程序员们每年至少学习一门新语言，主要是为了让我们去学习新的编程模型，而不提供新编程模型的语言不值得刻意去学习。

不过，这就需要你对程序设计语言有着更深的理解。下一讲，我们来看程序设计语言的接口，看看更具体的语言演化是如何发生的。

如果今天的内容你只能记住一件事，那请记住：**每年至少学习一门能够提供新编程模型的程序设计语言。**



极客时间

思考题

最后，我想请你分享一下，你最近打算学习哪门新的程序设计语言呢？为什么？欢迎在留言区分享你的想法。

感谢阅读，如果你觉得这一讲的内容对你有帮助的话，也欢迎把它分享给你的朋友。

精选留言：

- 业余爱好者 2020-06-10 01:00:37
  操作系统等各种软件有模型，编程语言也有模型，计算机世界里模型真是无处不在啊。模型就是一种看待世界的方式，一种模型的提出的最大的动力是为了解决某个问题。

  编程语言的发展就是各种编程模型的演化。从图灵机模型到冯诺依曼的存储程序模型，机器语言，汇编，一直到千姿百态的高级语言。每种语言都提供了一个编程模型，这个模型越来越高级，越人性化，越贴近人的思维。

  拿自己熟悉的java语言举例。尽管java是用c++语言写成的，但Java提供了不同于c++的内存与线程模型（Java的六种线程状态并不与os的各种线程状态一一对应，内存的划分也是如此），大大简化了开发。

  每年学一门新语言，也就是每年学一个新的编程模型。 [6赞]

  作者回复2020-06-11 20:30:46
  很好的补充，我这里只说了语言的演变，你补充的线程信息很好！

- 北天魔狼 2020-06-10 06:43:16
  非科班PHPer，今年目标拿下C语言
  参考书是《深入理解计算机系统》，我看不懂，所以我又找了《计算机是怎样跑起来的》，《程序是怎样跑起来的》，《网络是怎样连接的》，《图解HTTP》,《图解TCP/IP》,现在就剩程序和网络那两本没看，这个月看完。下个月开始死磕《深入理解计算机系统》 [2赞]

  作者回复2020-06-10 15:29:44
  嗯，C 语言是弄懂计算机结构的好入口。《深入理解计算机系统》是非常好的书，不过，确实不容易读。

- qinsi 2020-06-11 18:22:39

两本《七周七语言》里包含的语言:

Clojure, Haskell, Io, Prolog, Scala, Erlang, and Ruby
Lua, Factor, Elixir, Elm, Julia, MiniKanren, and Idris

几年前因为感到规模化的js工程需要类型，调研过Elm、BuckleScript和TypeScript，今天看来赢家不言而喻

[1赞]

作者回复2020-06-11 20:22:43
能被记住的就是赢家。

- Jxin 2020-06-10 11:23:33
1.最早了解到LISP,应该是在Paul Graham的Hacker News和Arc语言（为了证明Arc写了Hacker News，Arc没火Hacker News却火了）。
2.Paul Graham提出了著名的 Blub 论断，这个论断讨论这样一个问题：“不同的编程语言到底有什么不一样？是不是一种编程语言比另外一种编程语言更高级？”
3.为了表达观点，他引入了一个叫作 Blub 的语言。这个语言比机器上的二进制语言要高级一点，但是比市面上的任何高级语言都要低级。为什么从一个使用 Blub 语言的程序员的角度去看，机器上的二进制语言是一个更加低级的编程语言？因为它缺乏了 Blub 拥有的语言特性。但是从一个使用二进制编程语言的人的角度来看，其实两者没什么差别，因为即便有多余的特性，二进制语言编程者也不会去用那些东西，而只会用属于二进制语言的那个子集。
4.结论就是: 使用更高级语言的人知道低级语言缺乏了一些高级语言的特性，所以低级语言显得更原始，而低级语言的开发者却无法发现高级语言里面蕴含的额外的语言特性。
5.我的认知: 个人不喜欢高低之分,毕竟存在即合理,每款现存的语言都会有它高的地方和低的地方。但是Paul Graham的论断其实也反应了一些问题。就是人们往往会陷于自己的已有认知(毕竟阻力最小)。做为一个java程序员,无论是写py还是写go多多少少都有点用py和go语法写java代码的味道(也因为我在py和go的水平不够)。所以栏主所说的，学习语言是增量学习这个是对的，但已有认知也会是陷阱，这就导致我可以很快学会一门语言的语法，但很难接受其背后的编程模型。已有认知带来的偏向性会让人对新的编程模型带有抵触的情绪。外在表现就是,喜欢在其他语言中找不足来满足自己内心的偏向心理。

6.我们这次专栏是软件设计之美，重在设计。深入语言这种，感觉背景比设计理论都要重。ruby和这章感觉要理解栏主表达的意思都显得比较吃力。是只有我能力不足吗？栏主可有什么书籍推荐？ [1赞]

作者回复2020-06-10 15:44:27
感谢你做出这么棒的分享！

在内容中说了，之所以要讲语言就是为了打破语言局限，在后面的内容中，你就会看到，有用 C 语言实现面向对象的，有用 Java 实现函数式编程的。做设计，拥有各种设计思想固然重要，但更重要的是，能够把它实现出来。

慢慢来，这是一个系列的内容，等这几篇之后，你再有不理解的话，我们可以来好好讨论一下。

- 六一王 2020-06-10 02:04:24
我是做前端开发的，有往全栈方向发展的打算，文中说的一种新语言的编程模型，那我选择最近学习 typescript 算是一种新语言么？ [1赞]

作者回复2020-06-10 08:55:52
如果你只用过JavaScript，那TypeScript值得一学，它提供了更好的程序组织方式。

- Harry 2020-06-12 08:36:17

  看过此文才知道，自己还在从 0 到 1 的阶段。

- 陈政璋 2020-06-12 07:43:47
  看过很多文章都提到过编程模型，一直没理解编程模型到底指的是什么。

- 张逃逃 2020-06-10 11:57:56
  js，前端后端跨平台全搞定

  作者回复2020-06-10 19:23:11
  理想很丰满，现实很骨感。

- escray 2020-06-10 09:39:09
  项目经验比较少，所以反而不太受语言的限制，C#、Java、Ruby 都能写一些，Python、JavaScript 也都接触过，不过缺点就是每一门语言都不是很精通。

  看过《程序员修炼之道》，对于每年学一门新语言也很向往，之前还考虑过挑战一下《七周七语言》。但是确实没有仔细考虑过为什么要学，以及如何学得更有效率。

  发现文章在描述语言发展史的时候没有提到 C#、JavaScript 和 Go，当然还有一些其他的语言。C# 应该和 Java 比较类似，JavaScript 接近动态语言，但是因为 V8 引擎，又有些不太一样，Go 语言我就不知道该如何定位了，好像是结余 C 语言和 Java 之间。

  之前并不喜欢 Java（其实是因为我没有看到 Java 最近几年的发展），但是发现在工作市场上，Java 的机会远多于 C# 和 Ruby。

  每一种语言（或者框架）的出现，其实都是试图解决某种问题，如果能丰富一下自己的“兵器库”，当然会带来软件设计功力的提升。

  那么问题来了，感觉一年时间（工作之余）其实不足以掌握一门编程语言，因为还要需要学习一些周边的框架和工具等等，这个问题怎么破？

  后面如果有时间，想要学习一下 Go 语言。

- 阳仔 2020-06-10 09:18:17
  作为一个老程序猿使用的编程语言有Java，Python，golang，kotlin。每种编程语言都有自己编程模型，一个编程语言都是为了解决现有的问题而出现的。从汇编语言到高级语言，从过程编程到面向对象编程，从线程模型到协程模型…都是一个不断进化发展的过程，也是编程思维的升级。建议程序猿每年学习一门新语言其实也为了理解编程语言的模型思维，从而更好的理解软件设计。

  作者回复2020-06-10 19:27:09
  非常好的补充！